Informe Tarea N°1

Lenguaje de programación y paradigmas

Profesor: Paulina Gonzales
Integrantes: José Pablo Bernal G,
Renato Galleguillos C,
Vicente Zapata T.

https://github.com/VicenteZapataT/App1

Diseño y Justificación de la solución

Estructuras de datos empleadas

El código se organizó alrededor de la idea de que la información encontrada dentro del *.csv* se guardaría en una estructura de dato, para luego ser leída desde ahí en el resto del código. La estructura de dato utilizada fue el struct, lo cual permitió almacenar datos de diferentes tipos, para así poder organizar y manejar los datos de manera eficiente. Es posible evidenciar la utilización de esta estructura dentro del programa, en el archivo denominado *structs.h* donde se define la estructura *Pizza*, que luego es utilizada para almacenar los contenidos del archivo *.csv*.

Justificación de la modularidad

Es conveniente utilizar el modularidad a la hora de programar ya que mejora el orden y eficiencia del código. Es por esto por lo que se dividió el programa en distintos archivos con objetivos especifico, lo cual permitió un mejor manejo de las funciones a realizar en cada módulo de código. El archivo *main.c* se limita a solo gestionar el flujo principal del programa, mientras que *util.c* solamente contienen las funciones auxiliares para el procesamiento de datos y análisis de estos datos. Por otro lado, los archivos *.h* se centran en declaraciones, en este caso de structs, funciones y constantes.

Uso de punteros a funciones

El uso de punteros en las funciones del programa permite una manipulación eficiente de datos y mejor gestión de memoria. En funciones como pms, dms, apo, ect, se pasan estructura y variables por referencia mediante punteros, lo cual evita copias innecesarias de información. Además, esto mejora el rendimiento del programa y facilita el retorno de resultados.

Interacción entre archivos

El archivo *main.c* actúa como controlador principal del programa. Desde aquí se gestiona la apertura del archivo *.csv*, y se llama a las distintas funciones como parseLine para interpretar su contenido. Estas funciones no se encuentran definidas en *main.c*, sino en el archivo *utils*, por lo que *main.c* incluye de encabezado *utils.h*, que declara todas las funciones auxiliares disponible para su uso en otros módulos.

La función parseline, definida en *utils.c* , se encarga de analizar por línea el archivo *.csv* y transformar su contenido en un struct. Este struct está definido en el archivo *struct.h*, que contiene la declaración del struct que se utiliza. Tanto *main.c* como *utils.c* incluye este encabezado para acceder al struct

El archivo *utils.c* tiene la implementación de todas las funciones, llevar el *.csv* a un struct, pms (pizza más vendida), junto con otras funciones que permiten distintos tipos de análisis, funciones que necesitan tener disponible los encabezados *struct.h* para el manejo de datos, y utils.h donde están inicializadas las funciones de este archivo.

Por último, el *Makefile* cumple la función de automatizar el proceso de compilación. Define reglas necesarias para compilar *main.c* y *utils.c*. de este modo, se asegura que todos los archivos interactúen adecuadamente para la construcción del programa.

Reflexiones Finales

Reflexión – Vicente Zapata T.

¿Qué fue lo más complejo o interesante de la tarea?

A mi parecer, lo más interesante de la tarea fue todo el proceso que conlleva programar en C, ya que este lenguaje es más complicado de instalar, ejecutar, y compilar, que otros más modernos, como Python. Por otro lado, la alta rigidez del lenguaje como la gestión de tanto las operaciones de reserva como las de liberación de memoria, que hay que explicitar en el código, es algo completamente novedosos para mí.

¿Cómo enfrentaron los errores, pruebas y debugging?

Dentro de todo, para enfrentar los errores y debugging, principalmente utilizamos StackOverflow, ChatGPT, y W3School. Estos recursos nos permitieron entender de forma más completa la forma correcta de trabajar con C, al igual que resolver errores comunes de sintaxis.

¿Qué lecciones aprendieron al implementar en C este tipo de lectura de archivos y cálculos de métricas?

Al implementar en C este tipo de lectura de archivos y cálculos de métricas, vi reflejado el aprendizaje especialmente en el proceso que conlleva el parseo del archivo *.csv*, ya que no era simplemente instalar una librería y asignarlo hacia un DataFrame, como se podría realizar en Python, sino que tener que realizar un código tanto como para la lectura de este, como para la estructura de datos en la cual sería almacenada, el Struct.

Reflexión – José Pablo Bernal.

¿Qué fue lo más complejo o interesante de la tarea?

Lo más complejo de la tarea a mi parecer fue el parseo del archivo CSV, especialmente la parte de los ingredientes, ya que las comillas causaban varios errores al momento de leer los datos. Después de investigar un poco, logre entender mejor el cómo funcionaba y se hizo más manejable. Lo que me pareció más interesante fue la parte inicial del programa, donde se requería generar las salidas en un orden especifico. Aunque solo se utilizaron condicionales simple como if, me pareció entretenido ordenar bien la lógica para que todo saliera como se esperaba.

¿Cómo enfrentaron los errores, pruebas y debugging?

Para enfrentar los errores, recurrí muchas veces a Stack Overflow para resolver dudas específicas. Sino no encontraba una respuesta útil, me apoyaba en ChatGPT para entender mejor el problema y buscar soluciones.

¿Qué lecciones aprendieron al implementar en C este tipo de lectura de archivos y cálculos de métricas?

Aprendí que parsear un archivo CSV en C y cargar sus datos a un struct puedes ser complejo, y siempre hay que tener en cuenta su formato. Por otra parte, el cálculo de métricas me resulto más simple, en mi caso, solo implicaba realizar operaciones básicas para obtener el resultado, pero lo más desafiante de esto, en mi opinión, fue pensar bien la lógica para recorrer los datos necesarios.

Reflexión – Renato Galleguillos.

¿Qué fue lo más complejo o interesante de la tarea?

Por mi parte, lo más complejo fue la instalación de C y C++ en Visual Studio Code, principalmente debido a que siempre trabajé con Python y con archivos sencillos, por lo que incluir nuevas formas de programar era nuevo para mí y fue mucho más complicado de lo que esperaba. Tuve que seguir las instrucciones de múltiples videos para identificar errores o diferencias con los videos guía, incluso llegué al punto de pedir ayuda a distintas IA's, como ChatGPT o DeepSeek.

¿Cómo enfrentaron los errores, pruebas y debugging?

Con respecto a la instalación de C/C++, finalmente, para ahorrarme vueltas, desinstalé lo que tenía de C y C++, desinstalé Python y cambié la localización de Visual Studio Code, luego reinstalé todo y pude continuar. Respecto a errores del resto de la tarea, recurrí principalmente a ChatGPT y Stack Overflow como medios de ayuda.

¿Qué lecciones aprendieron al implementar en C este tipo de lectura de archivos y cálculos de métricas?

Aprendí que no debo subestimar algo que puede parecer simple como manejar archivos CSV, pensé que sería algo sencillo, como lo es en Python que usas librerías y todo queda listo, en C hay un proceso mayor que requiere más atención y precisión.

Explicación del uso de IA

Principalmente la IA fue utilizada para la asistencia en el aprendizaje, es decir, la explicación de conceptos y resolución de dudas que surgieron durante la elaboración de la tarea. Por otro lado, también fue utilizada para guiar ciertas partes del código elaborado. Es decir que se

utilizó código generado por la IA, que como equipo fuimos capaz de comprender, y modificar para incorporar dentro del programa de forma efectiva. Un ejemplo de lo antes mencionado seria en el código que respecta el parseo. Dentro de este, el uso de IA nos permitió generar un código que almacenara los ingredientes dentro de un miembro, proceso en el cual se había tenido complicaciones, debido a la lectura de las comillas que engloban todos los ingredientes de la pizza.